(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
28 December 2000 (28.12.2000)

PCT

(10) International Publication Number
WO 00/79799 A2

(51) International Patent Classification[7]: H04N 7/24, G06F 17/30

(21) International Application Number: PCT/US00/17372

(22) International Filing Date: 23 June 2000 (23.06.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/338,614 23 June 1999 (23.06.1999) US
09/551,446 18 April 2000 (18.04.2000) US

(71) Applicant: SARNOFF CORPORATION [US/US]; 201 Washington Road, CN 5300, Princeton, NJ 08543 (US).

(72) Inventors: PEJHAN, Sassan; 4 Heritage Boulevard, Princeton, NJ 08540 (US). CHAI, Bing-Bing; 51-03 Fox Run Drive, Plainsboro, NJ 08536 (US). SODAGAR, Iraj; 3068-C Via Alicante, La Jolla, CA 92037 (US). FESTA, John; 206 Central Avenue, Metuchen, NJ 08840 (US).

(74) Agents: MOSER, Raymond, R. et al.; Thomason, Moser and Patterson LLP, 2-40 Bridge Avenue, P.O. Box 8160, Red Bank, NJ 07701 (US).

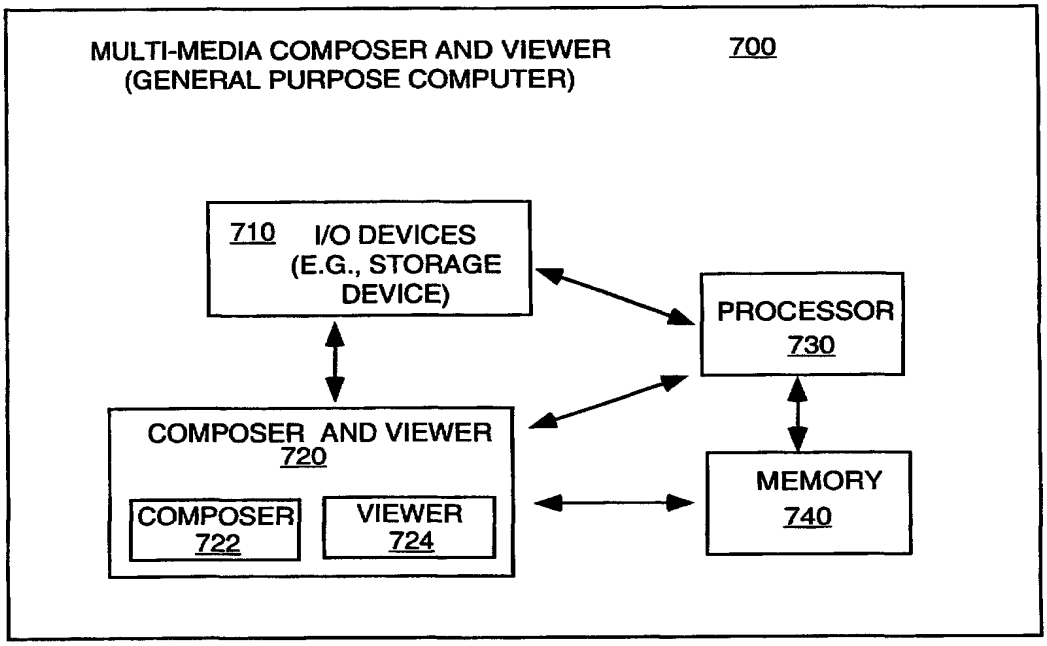
(81) Designated States *(national)*: CA, IL, JP, KR.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD AND APPARATUS FOR COMPOSING AND VIEWING IMAGE SEQUENCES



(57) Abstract: A method and apparatus (700) that provides authoring and viewing of multi-media contents. Namely, the invention provides an architecture for object-based multimedia authoring and composition.

WO 00/79799 A2

METHOD AND APPARATUS FOR COMPOSING AND VIEWING IMAGE SEQUENCES

This application is a continuation-in-part of U.S. Application serial number 09/338,614, filed June 23, 1999, which claims priority to U.S. provisional patent application serial number 60/103,762, filed October 9, 1998, both of which are herein incorporated by reference.

This invention was made with U.S. government support under contract number NMA 202-97-D-1003/0001. The U.S. government has certain rights in this invention.

The invention relates to multi-media authoring. More particularly, the invention relates to a method and apparatus for providing an architecture for object-based multimedia authoring and composition.

BACKGROUND OF THE INVENTION

As a result of the wide-spread use of powerful and multimedia friendly personal computers, it has become increasingly desirable to generate and view digital video clips. Video clips are becoming abundant on many INTERNET web sites and have been available on CD-ROMs for many years now. Unlike other traditional media, such as audio and text, video clips in their raw format can become prohibitively large computer files, consuming storage and bandwidth at unacceptably high rates. A substantial amount of research has therefore been performed over the past 30 years to develop efficient video compression algorithms. Several standards, including MPEG (-1, -2, -4), H.261 and H.263 have been developed. Almost all digital video sequences, whether on the web, on CD-ROMs or on local hard disks, are stored in one compressed format or another.

Given the ease with which video clips can be played back on a computer, there is a natural desire to have the same type of controls as one has with regular (analog) video players (such as Play, Stop/Pause, Fast Forward, Slow Motion, Reverse) as well as more sophisticated controls, such as random frame access, jumping to the beginning or end of a clip, or

jumping to the next or previous scene. Such controls can easily be implemented for raw video: the size of the frames are fixed, the position of each frame in the bitstream is known and frames can be accessed and displayed independently from one another. For compressed video, implementing some of these controls is challenging. Compressed frames have a variable size and their position in the bitstream may not be readily available. Moreover, if predictive coding is used (such as motion compensation), a given frame may not be decodable independently of other frames in the sequence. The operations which are simple to implement for compressed streams include Play, Stop/Pause, Slow Motion and Rewind, which are currently performed by most standard software decoders. The challenging operations include random frame access, fast forward, playing in reverse and jumping to the next scene change.

Brute force solutions to these challenges could be implemented using a very powerful computer, or when dealing with very small resolution and/or relatively short sequences. For instance, the Fast Forward control could be implemented by decoding and displaying the clip at two or three times the natural speed. With high resolutions and long sequences, however, this is not a practical option, particularly in cases where the video is being streamed over a network.

Additionally, as multi-media contents proliferate, it is advantageous if an authoring or composing tool is available that provides an efficient and user friendly graphical interface for composing a multi-media scene. Namely, various multimedia "objects" (e.g., video clips, audio clips, still images, animation and the likes) may exist and are stored in a storage medium. It would be advantageous to have functions that will allow a user to easily select and integrate these objects from an object library into a multi-media scene. For example, VCR-type controls as discussed below can be adapted as additional functions into such authoring or composing tool.

Therefore, there is a need in the art for a method and apparatus of providing VCR-type controls to a compressed video bitstream, and for a method and apparatus for composing a multi-media scene.

SUMMARY OF THE INVENTION

One embodiment of the present invention is a method and apparatus for multi-media authoring. The invention relates to a method and apparatus for providing an architecture for object-based multimedia authoring and composition.

One aspect of the present invention is that the architecture of the multi-media composer allows for automatic addition of a newly added multi-media object into the existing scene description. Namely, when a multi-media object is inserted into a scene composition area of the multi-media composer, relevant information associated with the multi-media object is automatically added to an existing scene description (or a new scene description will be created if no scene description currently exists). Various types of object oriented (OO) programming "objects" are created to implement this feature.

Additionally, other features or functions within a composed scene are supported by the present architecture of the multi-media composer. These functions, include but are not limited to, resizing and positioning of multi-media objects; organizing z-order of multi-media objects, implementing background transparency of multi-media objects, and providing scene playback of multi-media objects. Thus, the present invention discloses a novel and useful multi-media composer that can be employed to efficiently generate multi-media contents.

BRIEF DESCRIPTION OF THE DRAWINGS

The teachings of the present in vention can be readily understood by considering the following detailed description in conjunction with the accompanying drawings, in which:

FIG. 1 depicts a block diagram of a video sequence encoder in accordance with the present invention that produces a compressed btstream and an associated auxiliary file;

FIG. 2 depicts a file structure for a first embodiment of an auxiliary file;

FIG. 3 depicts a file structure for a second embodiment of an auxiliary file;

FIG. 4 depicts a file structure for a thir d embodiment of an auxiliary file;

FIG. 5 depicts a block diagram of a decoder for decoding bitstreams produced by the encoder of FIG. 1;

FIG. 6 depicts a block diagram of a client and server for streaming, decoding, and displaying remote bitstreams produced by the encoder of FIG. 1;

FIG. 7 depicts a block diagram of a multi-media composer and viewer of the present invention;

FIG. 8 depicts a view of the multi-media composer and viewer of the present invention on a display;

FIG. 9 depicts a flowchart of a method for composing a multi-media scene using object oriented programming elements;

FIG. 10 depicts a flowchart of a method for resizing and positioning one or more multi-media objects within a composition scene of the present invention;

FIG. 11 depicts a flowchart of a method for organizing the z-order of one or more multi-media objects within a composition scene of the present invention;

FIG. 12 depicts a flowchart of a method for implementing background transparency of one or more multi-media objects within a composition scene of the present invention; and

FIG. 13 depicts a flowchart of a method for implementing scene playback of one or more multi-media objects within a composition scene of the present invention.

To facilitate understanding, id entical reference numerals have been used, where possible, to designate identical elements that are common to the figures.

DETAILED DESCRIPTION

The major video coding techniques/standards in use today include H.263, geared towards low bit-rate video, MPEG-1, developed for CD-ROM applications at around 1.5 Mbits/s and MPEG-2, designed for very high quality video (HDTV) at around 10 Mbits/s. Although there are major

differences among these three standards, they are all based on the same basic principles described below.

Each frame of video can be one of three types: Intra-coded (I) frames (i.e., anchor frames), Predicted (P) frames and Bi-directionally predicted (B) frames. The I-frames are encoded very much like still images (i.e., JPEG) and achieve compression by reducing spatial redundancy: a Discrete Cosine Transform (DCT) operation is applied to 8x8 blocks of pixels within the frame, starting from the top, left block and moving to the right and down the rows of pixels. To complete the encoding of an I-frame, the DCT coefficients are then quantized and entropy encoded.

The P-frames are predicted from a preceding I- or P-frame. Using motion estimation techniques, each 16x16 MacroBlock (MB) in a P-frame is matched to the closest MB of the frame from which it is to be predicted. The difference between the two MBs is then computed and encoded, along with the motion vectors. As such, both temporal and spatial redundancy is reduced. B-frames are coded in a manner similar to P-frames except that B-frames are predicted from both past and future I- or P-frames.

I-frames are much larger than P or B frames, but they have the advantage of being decodable independent of other frames. P and B frames achieve higher compression ratios, but they depend on the availability of other frames in order to be decoded.

The first embodiment of the invention for implementing VCR type controls generates a small, separate auxiliary 'vcr' file for each compressed video sequence (bitstream). This auxiliary file contains key information about the associated bitstream that enables efficient implementation of VCR type controls. Specifically, for each compressed video stream there is an associated auxiliary file (e.g., with the same prefix as the compressed file name but with a `vcr' suffix). This auxiliary file primarily contains information about the position of Intra-coded frames (I-Frames) within the compressed bitstream.

Fig. 1 depicts a block diagram of a video sequence encoder system 100 containing an encoder 102, an auxiliary file generator 104 and a storage device 108 that operate in accordance with the present invention.

The encoder 102 encodes a video sequence in a conventional manner (e.g., as described above), but also produces I-frame information for use by

the auxiliary file generator 104. This information pertains to the location of the I-frames within the encoded bitstream 110, e.g. the position of the I-Frame with respect to the beginning (or end) of the bitstream. The auxiliary file generator 104 produces an auxiliary file 106 for each encoded bitstream 110.

Video sequences may be encoded at either a variable or constant frame rate. The former may occur when encoders drop frames, in an irregular fashion, in order to achieve a constant bit-rate. Furthermore, even if the frame rate is constant, I-Frames may or may not occur at fixed intervals. Such aspects of the coding process are not specified by the standards but are left to implementers. For some applications, it may make sense to insert I-Frames at fixed intervals (e.g., every 30th frame can be an I-Frame). For other applications, implementers may decide to insert an I-Frame only whenever there is a scene change - something which may occur at irregular time intervals. The auxiliary file has a different format depending on whether I-Frames are inserted at fixed or variable intervals.

When the I-frames are contained at fixed intervals, i.e., the I-frames are generated at a fixed interval by the encoder 102, the auxiliary file 106 has a particular form that facilitates efficient implementation of the invention. FIG. 2 illustrates the format of an auxiliary file 200 for use with a bitstream having a fixed I-frame interval. The auxiliary file 200 contains a field 202, e.g., one byte, at the head of the file indicating the size of the fixed interval. Next, a field 204, e.g., four bytes for every I-frame is included in the header to indicate the offset from the beginning (or the end) of the bitstream at which each I-frame is located.

If the interval between I-Frames is variable, the auxiliary file 200 of FIG. 2 is augmented with additional information to become the auxiliary file 300 of FIG. 3. The first field 302 of auxiliary file 300 is still that of the I-frame interval, but the field value is set to 0 (or some other special code) to indicate a variable frame rate. There will now be 2 fields per I-Frame: Field 306 containing a 2-byte frame number and field 308 containing the 4-byte offset information. Field 304, which indicates the total number of frames in the entire sequence, can be optionally added to the auxiliary file 300 (placed right after the frame interval field 302). As will be described below, this

optional information can help speed up the implementation of the random frame access control.

Finally, a one-bit Scene Change Indicator (SCI) field can be inserted in the auxiliary file for each I-Frame, indicating whether there has been a scene change or not from the previous I-Frame. One way of inserting this field is to add another one-byte field 310 for each I-frame, with the first bit serving as the SCI and the other bits reserved for future use. Alternatively, the first bit of the 4-byte offset field 308 can be designated as the SCI field 312, with the remaining 31 bits used for the offset, as shown in FIG. 4.

Since the file format 300 for variable I-frame intervals is a superset of the one for fixed I-frame intervals (format 200), it could be used for both cases. This makes the implementation of the invention slightly easier. The additional 2-bytes per I-frame will make the auxiliary files larger, however, for the case of fixed I-frame interval. Whether the trade-off is worth it or not is a choice for implementers to make and will vary from one case to another. All in all, however, the size of the auxiliary files generated is negligible compared to the size of the compressed video file. For the fixed I-frame interval case, the size is basically four bytes multiplied by the number of I-frames. If I-frames are inserted as frequently as even three times a second (i.e. once every tenth frame), then the auxiliary file adds 12 bytes (84 bits) per second. Even for a very low bit-rate sequence (say 5 kbit/s) the additional storage required for the auxiliary file is negligible. For the case of variable I-frame interval, the size of the auxiliary file is approximately six bytes multiplied by the number of I-frames. That translates into 126 bits/s, assuming three I-frames per second on the average.

FIG. 5 and FIG. 6 depict block diagrams of two different systems ("players") for playback of compressed video bitstreams with VCR-type controls. FIG. 5 depicts a player 500 that operates to playback locally stored video files. This player 500 comprises a User Interface/Display 502, a decoder 504, an auxiliary file processor 506 and local storage 108. The user interacts with the system through the user interface (e.g., a graphical user interface that has various "buttons" for VCR controls). The decoded bitstream may also be displayed here or on a separate display. The decoder 504 operates like any standard decoder, except that when VCR commands are issued, it interacts with the auxiliary file processor 506 to determine the

location in the bitstream from where the decoder needs to start decoding. The auxiliary file processor 506 in turn retrieves that information from the auxiliary file. Both the bitstream and the associated auxiliary file are stored locally on the storage device 108.

FIG. 6 depicts a system 600 where the bitstream and associated auxiliary file are stored remotely on a server 602. The bitstream is streamed to the player 601 over a network 612. When the user Interface/Display 602 processes a VCR command issued by the user, the decoder 604 relays this command over the network 612 to the server 602. A buffer 610 located between the network 612 and the decoder 604. The server 602 then interacts with the auxiliary file processor 606, which now resides on the server 602, to determine the location within the bitstream from which the server should start transmission.

When a user (client) requests to view a random frame within a compressed video bitstream (i.e., the user requests random frame access) within either system 500 or 600 , there are three cases to consider:

1. When the frame to be dec oded (selected frame) is the one after the last decoded frame, this is the normal video sequence play mode for the systems 500 and 600. Thus, the decoder 504 or 604 operates in a conventional manner without needing to retrieve any information from the auxiliary file, i.e., the decoder sequentially selects frames for decoding and display.

2. When the frame to be decoded (selected frame) is sometime in the future relative to the frame that is presently being decoded, but before the next I-frame, the system 500 or 600 needs to decode frames as in the usual play mode using the interframe predictions but without displaying the decoded frames until the desired frame is reached. As such, the decoder 504/604 blocks display of the decoded frames until the selected frame is decoded.

3. All other cases require special handling. First the I-frame prior to the selected frame has to be identified, i.e., the I-frame prior to the selected frame must be identified when given the current frame number being decoded and given the fact that the first frame in the sequence is an I-frame. If the I-frame interval is fixed, the selected frame is easily determined. Next, the offset of the I-frame will be read from the auxiliary file 200 and

provided to the decoder 504/604. Since there is a fixed size for the auxiliary file header and a fixed sized field (4-byte field 204) for each I-frame, determining the offset is trivial. The bitstream pointer that selects frames for decoding in the decoder 504/604 would then be moved according to the offset retrieved from the auxiliary file. The I-frame and the subsequent P-frames would be decoded but not displayed until the selected frame is decoded.

When the compressed bitstream is stored remotely (as in FIG. 6), there will be no changes for scenarios one and two above since all frames have to be retrieved and decoded though not displayed sequentially. For scenario three, once the system 600 determines the last I-frame prior to the frame of interest, the system 600 sends a stop message to the server 602 supplying the bitstream, only by a request for the I-frame mentioned. The server 602 then looks up the offset of that I-frame in the auxiliary file at the server 602, reset the pointer to the compressed bitstream file, and start transmitting bits from that point. The rest of the decoding process remains as described above.

When a variable I-frame interval is used, as before, when a frame needs to be decoded, one of the three cases listed above would apply. For the first case (decoding the next frame) there is no difference. For the second and third cases, however, the decoder has to determine if there is an I-Frame which preceded the frame of interest or not. To this end, it has to look up the 2-byte frame numbers (field 306) in the auxiliary file 300 and extract the appropriate I-Frame accordingly.

To speed up the search for the appropriate I-Frame, the field 304 indicating the total number of frames in the entire sequence is used. As such, the server 602 compares the number of the frame to be decoded with the total number of frames in the sequence and determines an estimate of where in the auxiliary file the server wants to start the I-frame search.

The scenarios described above apply to both local and remote files. The only difference occurs in the third case: for local files, the player (client) 500 has to perform the tasks indicated; for remote files, the client 601 sends a request to a server 602 for a random frame, and the server 602 has to look up the auxiliary file and resume transmission from the appropriate place in the bitstream.

When a user requests a jump to the next or previous scene, the auxiliary file will be scanned to find the next/previous I-Frame that has the scene change bit set to TRUE. That frame is decoded and displayed and the clip starts playing from that point onwards. Algorithms for detecting scene changes are well-known in the art. An algorithm that is representative of the state of the art is disclosed in Shen et al., "A Fast Algorithm for Video Parsing Using MPEG Compressed Sequences", International Conference on Image Processing, Vol. 2, pp. 252-255, October 1995.

In the present invention the auxiliary file information is used to provide a fast forward effect in the decoded video. The Fast Forward operation can simply be viewed as a special case of random access. Running a video clip at, say, three times its natural speed by skipping two out of every three frames is equivalent to continuously making 'random' requests for every third frame (i.e., requesting frame 0, 3, 6, 9, and so on). Every time a frame is requested, the random frame access operation described above first determines the position of the nearest preceding I-frame just as before. The frames from that I-frame to the selected frame are decoded but not displayed. As such, only the requested frames are displayed, i.e., every third frame.

In addition to frame jumping and fast forward, the invention includes two embodiments for implementing Reverse play (both normal and fast speed). The first embodiment, which is simpler but less efficient, is to view the reverse play as a special case of random frame access. For each frame, the server or local decoder invokes the random frame access mechanism described above. The number of the 'random' frame to be retrieved is decremented by one or N each time, depending on the playback speed. This scheme is inefficient due to the existence of predicted frames. To see why, consider the following case:

Assume a sequence where every 10 th frame (0,10,20, and so on) is an I-Frame, and all other frames are forward predicted (P-) frames. To play this video clip in reverse, using the random access method, some of the frames have to be decoded several times. For example, after frame 10 is decoded and displayed, frames 0 through 8 are decoded (but not displayed) so that frame 9 can be decoded and displayed. Then, frames 0 through 8 are decoded again so that frame 8 can be displayed, and so forth. As such, the

same frames must be decoded over and over to achieve a sequence of frames to be displayed in reverse order. The need for repetitively decoding the same frames is very inefficient.

The second embodiment involves caching in memory (cache 508 or 608) all the frames in a Group of Pictures (GOP) when the Reverse control is invoked. Thus, in the example above, after frame 10 (an I-Frame) has been decoded and displayed, the decoder 504/604 can decode all frames between 0 and 9, cache them in memory (508/608 in FIGS. 5 and 6), and then display them in reverse order (from 9 to 0). While this would be much more efficient than the first embodiment, this embodiment does have the drawback of consuming significant amounts of memory, if the GOP is large and/or if the resolution of the video is high.

FIG. 7 depicts a block diagram of a multi-media composer and viewer 700 of the present invention. In one embodiment, the multi-media composer and viewer 700 can be tailored to handle MPEG compliant multi-media contents (e.g., MPEG-4 and the likes), thereby earning the name "MPEG Authoring Tool" (MAT). Although the multi-media composer and viewer 700 is described below in view of MPEG-4 multi-media contents, it should be understood that the present invention is not so limited and can be adapted to other multi-media contents that are compliant to other standards, such as MPEG-2, MPEG-7, JPEG, ATSC, H.263 and the likes.

The multi-media composer and viewer 700 provides both composing as well as viewing functions and capabilities. Thus, the viewing capabilities as described above can be deployed in the present multi-media composer and viewer 700, whereas composing functions and capabilities are now described below.

The MPEG Authoring Tool (MAT) provides a graphical interface for the composition of MPEG-4 scene descriptions as well as a facility for viewing the playback of the new scene. Various elements or features of graphical user interface design are employed, e.g., point-and-click selection, positioning, resizing, and z-order positioning of multi-media objects in the scene.

For example, FIG. 8 depicts a view 800 of the multi-media composer and viewer of the present invention on a display, e.g., a computer screen. To illustrate, one such element is the ability to drag-and-drop a file, e.g., from

Windows Explorer (a trademark of Microsoft Corporation of Redmond, Washington) onto the scene composition area 810 using a thumbnail bar 820 having a plurality of selectable multi-media objects 822-828. The scene composition area 810 also contains a selection status area 812 that displays various specific information relating to a particular multi-media object, e.g., its relative location in the scene composition area 810, its width, its height, its display order ("Z-order") and the likes. While drag-and-drop is a common feature, in general, the usual result of a drag-and-drop operation is only to open a new "document window" within the parent application. In the case of the present MAT, the result of the drag-and-drop operation results in the automatic addition of a new multi-media object into the existing scene description.

Specifically, for an authoring tool such as the MAT, where a complex multimedia scene can be composed using elementary audio, visual or text objects, there needs to be a mechanism to describe the composed scene. Thus, a scene description mechanism is needed and described below. In a software implementation, each object is represented by an instance of a media class (e.g., a specific video object can be represented as an instance of a Video Object Class). As such, it includes information regarding the position, scale, Z-order, time it appears in, or disappears from, the scene, etc. of that specific object. The scene description mechanism aggregates all of that information (for all the objects in the scene) into one single "Scene Description" file.

An example of such a file is an MPEG-4 BInary Format for Scenes (BIFS). Every time a new object is added to a scene, (or existing attributes of an existing object within a scene, such as position, size or z-order are modified), the scene description file is also modified to account for the newly added object (or changes to the existing object).

To illustrate, in order to generate a BIFS file, the present invention can cycle through all the objects in the composed scene, thereby collecting their attributes and writing them into a scene description file using the BIFS syntax. The update to the scene description file can occur immediately after an object is deposited into the scene composition area or when the object is modified within the scene composition area.

Alternatively, the update mechanism can be implemented in accordance to a schedule, e.g., every 15 minutes or when the user saves the composed scene.

The multi-media composer and viewer of the pr esent invention supports various types of multi-media objects, e.g., MPEG-4 object-based video, H.263 video, raw YUV-encoded video, an audio stream, and VTC still images. In the preferred embodiment, the MAT employs object-oriented (OO) design principles to achieve source code modularity and design extensibility for future additions. However, there is no design limitation, for instance, that would prohibit straightforward traditional software programming to implement a drag-and-drop feature of the present invention that would incorporate multi-media objects into the scene description.

FIG. 9 illustrates a flowchart of a method 900 for composing a scene using object oriented programming elements. Method 900 starts in step 905 and proceeds to step 910, where method 900 obtains a "media object" to be inserted into a scene composition area, e.g., executing a point-and-click operation.

Specifically, in the MAT, scenes are composed of one or more "media objects", each of which represents, for instance, a single multi-media object, e.g., an MPEG-4 video stream, audio stream, or VTC still image. In OO terminology, a media object would be called an "entity class" since it is responsible for maintaining state information about a real world thing. A media object contains information about the source of the multi-media object (e.g., the URL, or local file path) as well as information about the intrinsics of the data, such as image width, height, and color depth.

In step 920, method 900 generates a "rendered med ia object" associated with said media object. In order to render a multi-media object within the scene, a separate object called a "rendered media object" (another "entity class") is created which encapsulates knowledge of a single instance of the visual representation of the media object. The rendered media object contains information such as the scale at which the media object is drawn, spatial location of the object within the scene, and may also contain temporal information such as rendered frame rate (which may be different than the intrinsic frame rate).

In step 930, method 900 generates a "media object accessor" associated with said media object. Namely, the third object, "media object

accessor", is responsible for accessing the data referenced by the media object and providing it to the rendering subsystem for display. A media object accessor would be classified as a "boundary class" in OO terminology. The advantage of this third object is to encapsulate knowledge of how to access a media object from its source repository, and hide the details of disk or network access from the rest of the system. Media object accessors are responsible for maintaining state information such as the last accessed position in the video stream. Media object accessors are also responsible for raising an alert when the end of the stream is reached. Method 900 then ends in step 935.

It should be noted that if a media object is to be rendered in two locations within the scene, two rendered media objects are needed, both of which refer to a single media object. This arrangement results in a smaller memory footprint for the system, as well as reducing the need to transmit two copies of the same media object, thus potentially reducing transmission bandwidth requirements. This last advantage of reducing bandwidth requirements is the responsibility of the media object accessor, which can employ intelligent buffering. Although the broad description of the scene composition method is disclosed above, various additional functionalities are further described below.

Returning to FIG. 7, a block diagram of a multi-media composer and viewer of the present invention is illustrated. In one embodiment, the multi-media composer and viewer 700 is implemented using a general purpose computer. Thus, illustrative multi-media composer and viewer 700 comprises a processor (CPU) 730, a memory 740, e.g., random access memory (RAM), a composer and/or viewer 720, and various input/output devices 710, (e.g., a keyboard, a mouse, an audio recorder, a camera, a camcorder, a video monitor, any number of imaging devices or storage devices, including but not limited to, a tape drive, a floppy drive, a hard disk drive or a compact disk drive).

It should be understood that the composer 722 and viewer 724 can be implemented jointly or separately. Additionally, the composer 722 and viewer 724 can be physical devices that are coupled to the CPU 730 through a communication channel. Alternatively, the composer 722 and viewer 724 can be represented by one or more software applications (or even a

combination of software and hardware, e.g., using application specific integrated circuits (ASIC)), where the software is loaded from a storage medium, (e.g., a magnetic or optical drive or diskette) and operated by the CPU in the memory 740 of the computer. As such, the composer 722 and viewer 724 (including associated data structures) of the present invention can be stored on a computer readable medium, e.g., RAM memory, magnetic or optical drive or diskette and the like.

In operation, various multi-media information are stored and retrieved from a storage device 710 of the multi-media composer and viewer 700. The multimedia information may include, but is not limited to, various image sequences such as complete movies, movie clips or shots, advertising clips, music videos, audio streams, animation clips and the like. The image sequences may or may not include audio streams or data streams, e.g., closed captioning and the like. The multi-media composer and viewer is now described below with reference to various functions or features that are provided in the operation of scene composition.

Resizing & Positioning

An important feature of the present multi-media composer and viewer is its ability to provide resizing and positioning capabilities. The core task associated with graphically positioning and resizing multi-media objects in an apparatus and method such as the MAT is tracking and recording movements of a pointing device, e.g., mouse movements, known as "mouse-hit testing". A common technique for creating "hotspots" in a graphical image (e.g., each state in a map of the United States) involves overlaying invisible graphical "widgets" over each desired hotspot, and using the widgets' built-in mouse-hit testing facilities which are returned to the parent application via software "events".

To illustrate, FIG. 10 depicts a flowchart of a method 1000 for resizing and positioning one or more media objects within a composition scene of the present invention. Method 1000 starts in step 1005 and proceeds to step 1010 where method 1000 overlays each media object with a transparent widget.

Specifically, each media object rendered in the scene is overlaid by a transparent widget known as an "image control", which is responsible for

alerting the MAT which multi-media object the user has currently selected with the mouse. In step 1020, method 1000 then generates a separate "resizer" object that utilizes several such widgets in order to facilitate positioning and resizing. The unique architecture of this "resizer object" provides various novel feature of the present invention. Method 1000 then ends in step 1025.

In one embodiment, the "resizer" object is constructed of eight "label controls", one "image control", and one "shape control", all of which are standard graphical interface components provided with MS Visual Basic. Once a user selects an object in the scene, the resizer control is made visible and overlaid on the selected object. Once this is accomplished, all subsequent mouse events that occur within the region enclosed by the resizer control are captured by the resizer object. Visually, the resizer control consists of a transparent rectangular region overlaid on the selected object, along with eight "handles" which are smaller opaque boxes positioned at the corners and mid-points of each of the sides of the central rectangle, shown with reference numeral 814 in FIG. 8.

The central rectangle consists of an invisible image control, which is capable of receiving mouse events while maintaining transparency with its background, and a shape control that is needed to provide the user with a visual cue to the placement of the central rectangle. Two controls are needed here because while the image control provides mouse-tracking ability, it does not provide the feature rich visual outlining capabilities of the shape control, which, in turn, provides no mouse-tracking features. Each of the handles is composed of a label control, which like the image control provides mouse tracking, but unlike the image control, provides for good visual feedback to the user. The operation of the resizer control then is to capture the mouse events raised by these graphical elements, resize and reposition them according to the user's requests, and repackage those events into custom-generated events, alerting the parent software application of the resizer object that the size or position of the resizer graphical components have changed. It is then up to the parent application of the resizer object to resize/reposition the rendered media object currently associated with the resizer object, and update the data stored in the rendered media object accordingly.

Separating the resizer functionality into a separate object enable s the source code to be re-used across many different applications, thus reducing future development time. MPEG-4 BIFS scene description generation in the MAT is accomplished upon request of the user, by simply indexing through the list of rendered media objects in the scene, and passing size and position information from each rendered media object to the BIFS-generating subsystem.

To illustrate, for each rendered media object, information such as position (with respect to the display), scale, Z-order, frame rate and the like are maintained. Specifically, the MPEG-4 BIFs scene description uses its own VRML-like syntax to describe the objects in a scene and the relationships among them. When a user composes a scene and then "clicks" on a "generate BIFS" or similarly labeled button, then the MAT automatically translates the information from each "rendered media object" to BIFS syntax. It also generates the relationships using BIFS syntax.

Z-Order

Another important feature of the present multi-media composer and viewer is the positioning of multi-media objects within a so-called "Z-Order" or "back-to-front order". Namely, "Z-order" can be defined as the sequential overlay order of multi-media objects in the z-axis, i.e., the axis coming out of a screen. For example, if multi-media objects overlap, then the z-order defines the relative positioning of the multi-media objects, e.g., which multi-media object is in the front and which multi-media object is in the back and so on.

In brief, accomplishin g this ordering task requires two main functions. First, it is necessary to accurately determine which multi-media object is to be selected by the user as being in the front in the case where multiple multi-media objects overlap each other in space. The second task is to track programmatically which multi-media object is in front of which.

FIG. 11 depicts a flowchart of a method 1100 for organizing the z-order of one or more multi-media objects within a composition scene of the present invention. Method 1100 starts in step 1105 and proceeds to step 1110, where a z-order location is defined for a multi-media object. For

example, selecting a multi-media object to define its z-order, e.g., as being in the front of a plurality of media objects, and so on.

In step 1120, method 1100 shuffles the z-order of the multi-media object to its proper order. It should be noted that the difficulty in graphically positioning objects in the z-order lies in the Visual Basic limitation on setting the z-order of image controls. Setting the z-order is limited to positioning an image control at either the back or the front of the z-order, but "shuffling" an image control forward or backward one position is not supported.

To address this limitation, method 1100 is capabl e of providing this feature. Step 1120 involves shuffling an existing z-order of multi-media objects (if one already exists) such that the z-order of multi-media objects is at a point where the desired multi-media object should be positioned. The desired multi-media object is then inserted at this point in front or behind this location and the modified z-order is reshuffled forward or backward to arrive at a new z-order with the newly added multi-media object in the proper position. The following pseudo-code illustrates the process of shuffling an object back one position in the z-order:

```
Let z_index = z-position of the object to be shuffled.
Re-position object to be shuffled at the top of the z-order
For each z-position from the bottom of the z-order  to z_index – 2
Re-position object at z-position z_index – 1 to the top of the Z-order
```

It should be noted that the steps performed by method 1100 are not detectable by the user. Namely, in operation, a user will simply drag and drop a multi-media object from a thumbnail bar 820 onto the scene composition area 810. Upon detection of one of more existing multi-media objects at the desired location, the selection status area 812 will query what z-order is desired (e.g., 1,2 ... n) for the current selected multi-media object. The user can then type a z-order value into the selection status area 812. Alternatively, in one embodiment, the user can simply click on a second button on a multi-button mouse to have the selected multi-media object simply shuffle backward.

Returning to FIG. 11, in step 1130, method 1100 tracks the z-order of each media object in accordance with its rendered media object. Recall that each rendered media object is associated with an image control in order to accomplish mouse-hit testing as discussed above. Since Visual Basic provides a means to adjust the z-order of image controls, adjusting the z-order of each image control in alignment with its associated rendered media object is sufficient to distinguish which multi-media object is to be accepted by the user. This is valid since the image control at the top of the z-order will trap the mouse movement events, effectively shielding the image controls behind it. Tracking the z-order of each object programmatically is then a matter of maintaining an ordered list or array in which each element of the list is a reference to a single rendered media object, and ordinal position in which the reference appears in the list represents its position in the z-order. Shuffling the references in the list enables efficient management of z-order position tracking as discussed above.

Background Transparency

Another important feature of the present multi-media composer and viewer is the displaying or rendering of an image sequence with background transparency. Namely, rendering of object-based MPEG-4 encoded video allows that an object of arbitrary shape which is situated within a rectangular region (i.e., "bounding box") be rendered onto a background such that the area that is inside the bounding box but outside the arbitrarily shaped "object" region is transparent to the background. This is illustrated in FIG. 8 where multi-media object 826 is shown within a bounding box having an object of arbitrary shape 826b and a background 826a. It should be noted that a multi-media object may have multiple objects of arbitrary shape.

When the multi-media object 826 is selected and placed within the scene composition area 810, the composer of the scene can selectively elect to have the background 826a be made transparent. This feature allows the background of a selected multi-media object to be selectively replaced with another background.

Various techniques are available for accomplishing this effect, where these techniques often require both the object of arbitrary shape to be rendered, as well as, a second "negative" image, which is used as a mask. The original image and the mask together define the arbitrarily shaped "object" region as well as the transparent bounding box. In essence, the mask serves to negate or make transparent all other regions of the multi-media object with the exception of the arbitrarily shaped "object" region of interest. Thus, if a multi-media object contains multiple arbitrarily shaped "object" regions, then other masks can be generated such that only those selected "object" regions will be visible.

In addition to the multi-media object, its mask, and the destination image (i.e., background for the scene composition area 810), a fourth image is required in order to achieve an animated effect. This fourth image stores a copy of the destination image to be replaced by the object of arbitrary shape, so that it may be restored prior to moving the object of arbitrary shape to a new position on the screen.

FIG. 12 depicts a flowchart of an illustrative method 1200 for implementing background transparency of one or more multi-media objects within a composition scene of the present invention. Method 1200 starts in step 1205 and proceeds to step 1210, where method 1200 copies the region of the destination image to be overwritten to a temporary, off-screen buffer.

In step 1220, method 1200 renders the mask image onto the destination image stored in the buffer using a pixel-by-pixel logical "AND" operation, such that the mask creates a "hole" in the destination image to be filled by the object of arbitrary shape.

In step 1230, method 1200 renders the multi-media object image onto the destination image stored in the buffer by using a pixel-by-pixel "XOR" operation, such that the "object portion" (i.e., the object of arbitrary shape) of the multi-media object fills in the hole and the uniform non-object region surrounding the object. The "object portion" will reside inside the bounding box and will not corrupt the existing background, thereby giving the illusion of transparency.

In step 1240, the "composite" region stored within the buffer is then copied back onto the scene composition area, i.e., the composite region is returned to the destination image. Method 1200 then ends in step 1245.

In an alternate embodiment of the present invention, a fifth image buffer is required since the background of MPEG-4 object-encoded video often contains artifacts of the encoding process which must be removed prior to rendering the object onto the background. In this embodiment, the multi-media object image is first copied to an off-screen "staging" buffer and then combined with the inverse of the mask image using a logical AND operation to remove artifacts in the non-object region of the object image. The remainder of the technique is analogous to the technique already described above.

Scene Playback

Playback of a composed scene can be implemented by exploiting the same tools and objects that were used in composing the scene. In one embodiment, playback of a scene is accomplished using two separate objects.

FIG. 13 depicts a flowchart of a method 1300 for implementing scene playback of one or more multi-media objects within a composition scene of the present invention. Method 1300 starts in step 1305 and proceeds to step 1310 where a "media object renderer" object is generated for each rendered media object as discussed above. The "media object renderer" object is associated with each rendered media object and is responsible for rendering the media object in the scene according to the specifications in the rendered media object. Recall that rendered media objects are responsible for maintaining position and scaling information about a media object.

In step 1320, method 1300 generates a "media object player" object for each rendered media object as discussed above. This separate "media object player" object serves two purposes; it provides timing (i.e., rate control) of the playback for a single rendered media object, and provides an graphical interface to the user enabling interaction with the playback using VCR-style functions as discussed above. Separating the user interface/rate control functionality out from the rendering functionality primarily serves to facilitate playback of composite multi-media objects. For instance, the media object player object for the composed scene makes use of services provided by independent media object renderers for each object in the scene and simply coordinates the timing of the rendering of each object. Method 1300 then ends in step 1325.

Although various embodiments which incorporate the teachings of the present invention have been shown and described in detail herein, those skilled in the art can readily devise many other varied embodiments that still incorporate these teachings.

What is claimed is:

1. A method for composing a multi-media scene, said method comprising the steps of:

a) retrieving a multi-media object from a storage; and

b) depositing said retrieved multi-media object ont o a scene composition area, where said deposition of said retrieved multi-media object, causes an addition of information associated with said retrieved multi-media object into a scene description.

2. A method for resizing or positioning a multi-media ob ject within a multi-media scene, said method comprising the steps of:

a) overlaying a multi-media object with a transparent widget; and

b) generating a resizer object associated with said multi-media object.

3. A method for organizing a z-order, where a current multi-media object is inserted with a plurality of multi-media objects within a multi-media scene, said method comprising the steps of:

a) defining a z-order location for said current multi-media object;

b) shuffling said plurality of multi-med ia objects to allow insertion of said current multi-media object; and

c) tracking said z-order location of each of said plurality of multi-media objects in accordance with rendered media objects associated with said plurality of multi-media objects, where said rendered media objects contain information for rendering said multi-media objects.

4. A method for causing a region of a multi-media object within a multi-media scene to be made transparent, where said multi-media scene has a destination image and where said multi-media object has a background and at least one object of arbitrary shape, said method comprising the steps of:

a) storing a region of the destination image to a buffer;

b) rendering a mask of said multi-media object onto said stored destination image; and

c) rendering said multi-media object onto said stored destination image, thereby causing a region of said multi-media object to be made transparent.

5. A method for scene playback of a plurality of multi-media objects within a multi-media scene, said method comprising the steps of:

a) generating a media object renderer object for each of a plurality of rendered media objects, where each rendered media object contains information for rendering a multi-media object; and

b) generating a media object player object for each of said plurality of rendered media objects, where each media object player object contains timing information for rendering a multi-media object.

6. A computer-readable medium (710, 740) having stored thereon a plurality of instructions, the plurality of instructions including instructions which, when executed by a processor, cause the processor to perform the steps comprising of:

a) retrieving a multi-media object from a storage; and

b) depositing said retrieved m ulti-media object onto a scene composition area, where said deposition of said retrieved multi-media object, causes an addition of information associated with said retrieved multi-media object into a scene description.

7. A computer-readable medium (710, 740) having stored thereon a plurality of instructions, the plurality of instructions including instructions which, when executed by a processor, cause the processor to perform the steps comprising of:

a) overlaying a multi-media object with a transparen t widget; and

b) generating a resizer object associated with said multi-media object.

8. A computer-readable medium (710, 740) having stored thereon a plurality of instructions, the plurality of instructions including instructions which, when executed by a processor, cause the processor to perform the steps comprising of:

a) defining a z-order location for said current multi-media object;

b) shuffling said plurality of multi-media objects to allow insertion of said current multi-media object; and

c) tracking said z-order location of each of said plurality of multi-media objects in accordance with rendered media objects associated with said plurality of multi-media objects, where said rendered media objects contain information for rendering said multi-media objects.

9. A computer-readable medium (710, 740) having stored thereon a plurality of instructions, the plurality of instructions including instructions which, when executed by a processor, cause the processor to perform the steps comprising of:

a) storing a region of the destination image to a buffer;

b) rendering a mask of said multi-media object onto said stored destination image; and

c) rendering said multi-media object onto said stored destination image, thereby causing a region of said multi-media object to be made transparent.

10. A computer-readable medium (710, 740) having stored thereon a plurality of instructions, the plurality of instructions including instructions which, when executed by a processor, cause the processor to perform the steps comprising of:

a) generating a media object renderer object for each of a plurality of rendered media objects, where each rendered media object contains information for rendering a multi-media object; and

b) generating a media object player object for each of said plurality of rendered media objects, where each media object player object contains timing information for rendering a multi-media object.

11. Apparatus (700) for composing a multi-media scene, said apparatus comprising:

a storage (710, 740) for storing a plurality of multi-media objects; and

a composer (722), coupled to said storage, for retrieving at least one of said plurality of media objects onto a scene composition area, where said

retrieved multi-media object causes an addition of information associated with said retrieved multi-media object into a scene description.

12. Apparatus (700) for resizing and positioning a multi-media object within a multi-media scene, said apparatus comprising:

a storage (710, 740) for storing a plurality of multi-media objects; and

a composer (722), coupled to said storage, for receiving at least one of said plurality of media objects onto a scene composition area, and for sizing said at least one of said plurality of media objects by overlaying said at least one of said plurality of media objects with a transparent widget and generating a resizer object associated with said at least one of said plurality of media objects.

13. Apparatus (700) for organizing a z-order, where a current multi-media object is inserted with a plurality existing of multi-media objects within a multi-media scene, said apparatus comprising:

a storage (710, 740) for storing a plurality of multi-media objects; and

a composer (722), coupled to said storage, for receiving at least one of said plurality of media objects onto a scene composition area having a plurality of existing multi-media objects, and for defining a z-order location for said current multi-media object, and for shuffling said plurality of existing multi-media objects to allow insertion of said current multi-media object; and for tracking said z-order location of each of said plurality of multi-media objects within said scene composition area in accordance with rendered media objects associated with said plurality of multi-media objects, where said rendered media objects contain information for rendering said multi-media objects.

14. Apparatus (700) for causing a region of a multi-media object within a multi-media scene to be made transparent, where said multi-media scene has a destination image and where said multi-media object has a background and at least one object of arbitrary shape, said apparatus comprising:

a storage (710, 740) for storing a plurality of multi-media objects; and

a composer (722), coupled to said storage, for receiving at least one of said plurality of media objects onto a scene composition area, and for storing a region of the destination image to a buffer, and for rendering a mask of said multi-media object onto said stored destination image and for rendering said multi-media object onto said stored destination image, thereby causing a region of said multi-media object to be made transparent.

15. Apparatus (700) for scene playback of a plurality of multi-media objects within a multi-media scene, said apparatus comprising:

a storage (710, 740) for storing a plurality of multi-media objects; and

a composer (722), coupled to said storage, for receiving at least one of said plurality of media objects onto a scene composition area, and for generating a media object renderer object for each of a plurality of rendered media objects, where each rendered media object contains information for rendering a multi-media object, and for generating a media object player object for each of said plurality of rendered media objects, where each media object player object contains timing information for rendering a multi-media object.

FIG. 1

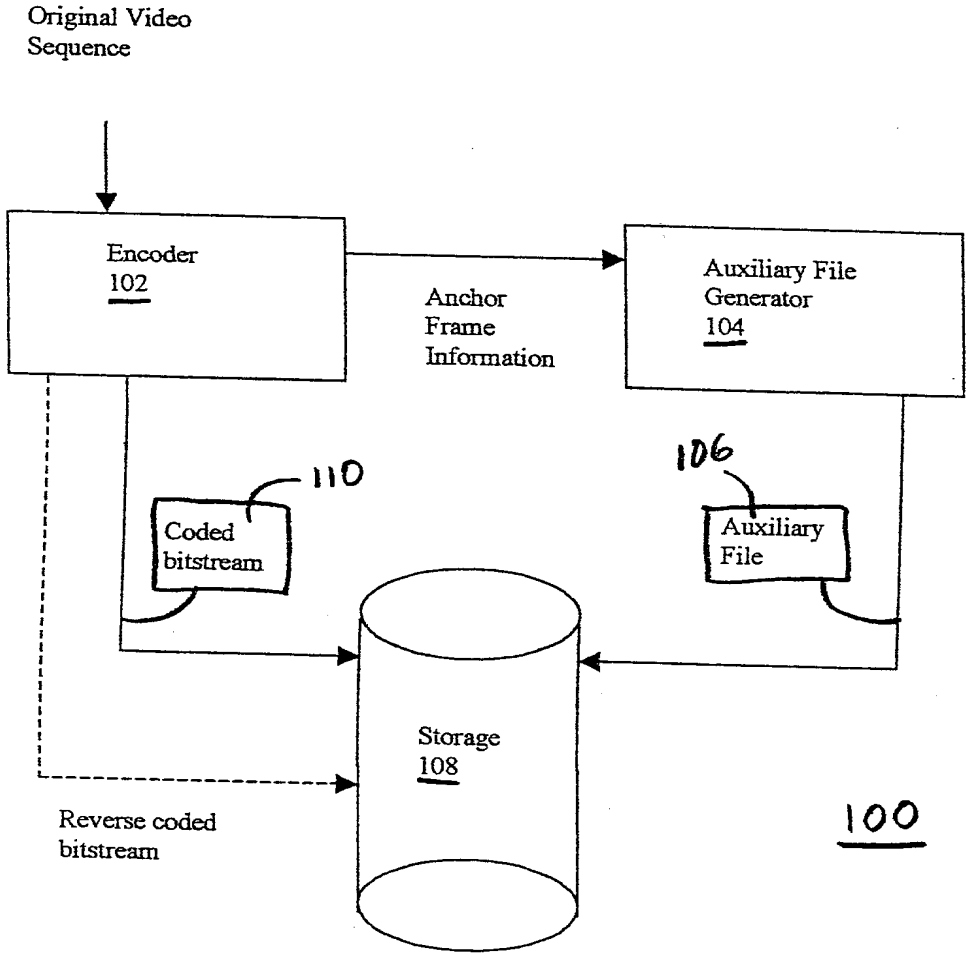

Original Video Sequence
Encoder 102
Anchor Frame Information
Auxiliary File Generator 104
110
Coded bitstream
106
Auxiliary File
Storage 108
Reverse coded bitstream
100

FIG. 2

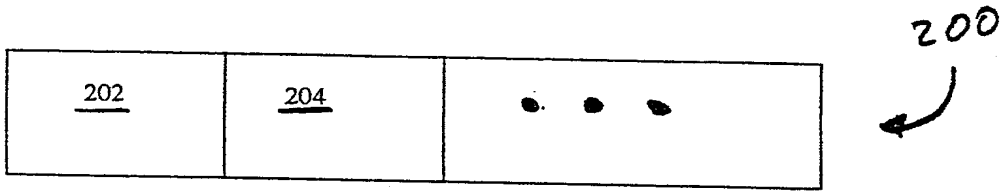


FIG. 3:

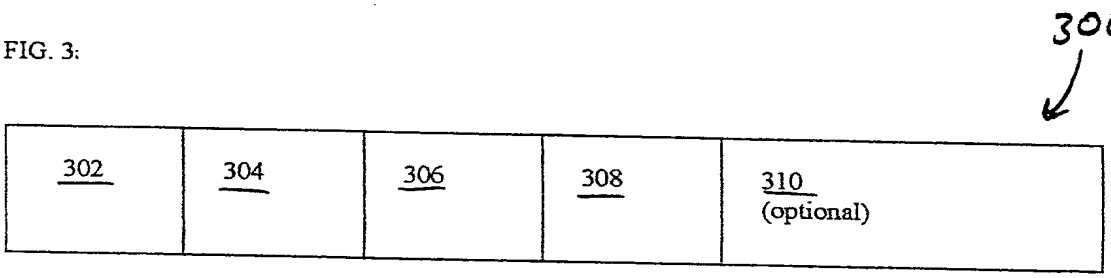


FIG. 4

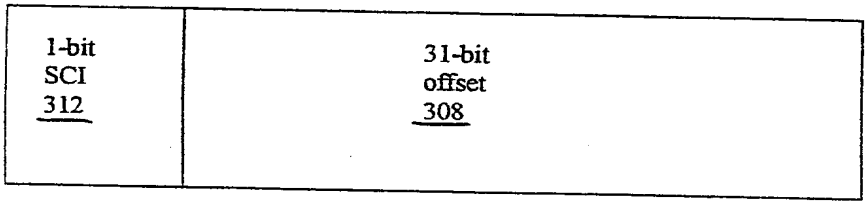


FIG. 5

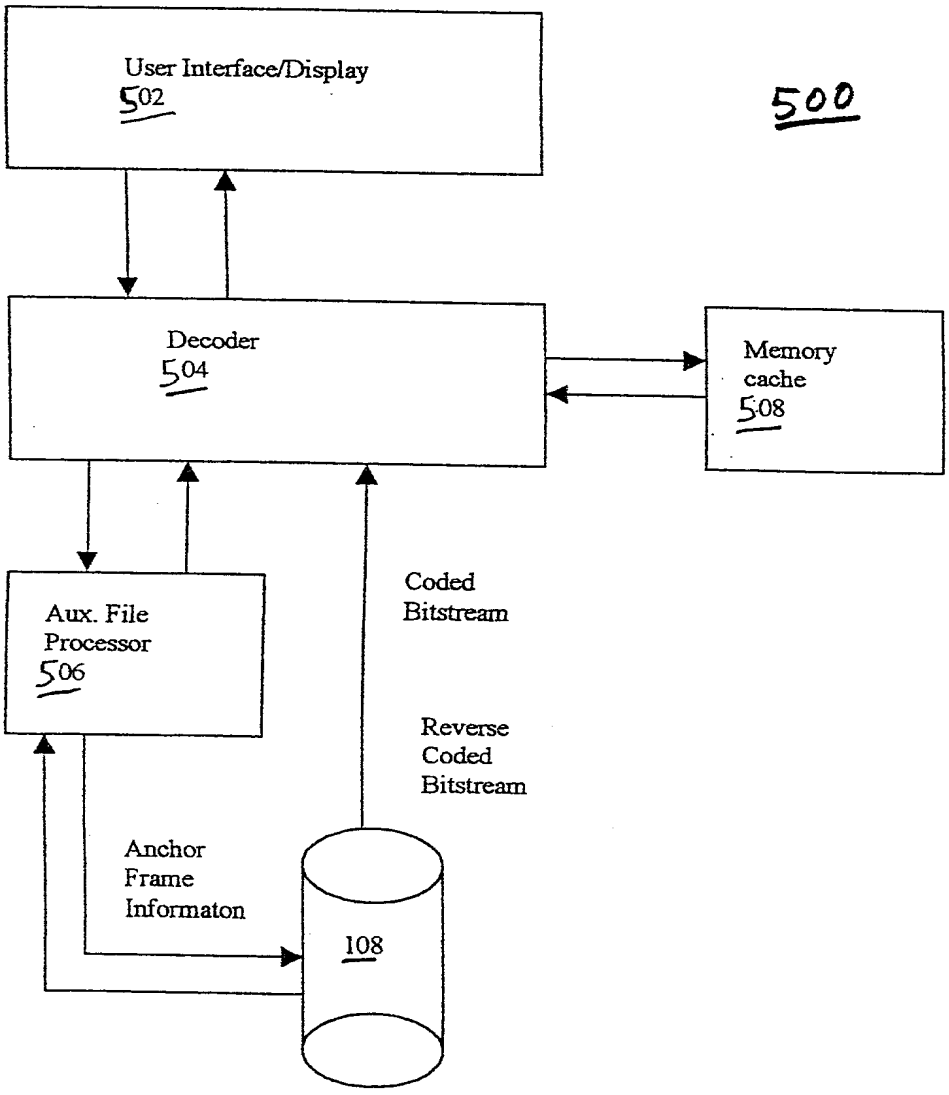
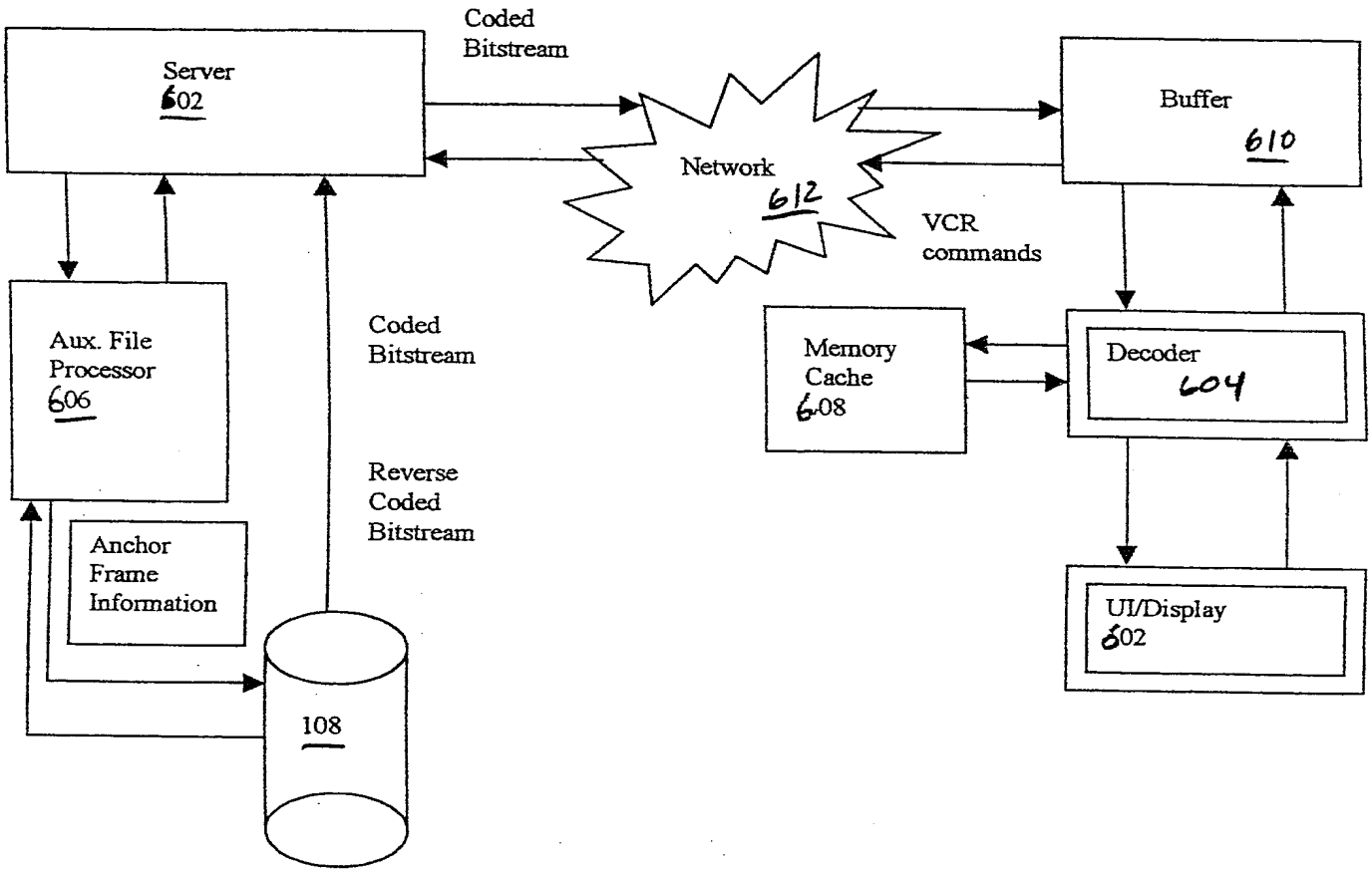

User Interface/Display
502
500
Decoder
504
Memory
cache
508
Aux. File
Processor
506
Coded
Bitstream
Reverse
Coded
Bitstream
Anchor
Frame
Informaton
108

FIG. 6


Server
602
Coded
Bitstream
Network
612
Buffer
610
VCR
commands
Aux. File
Processor
606
Coded
Bitstream
Memory
Cache
608
Decoder
604
Reverse
Coded
Bitstream
Anchor
Frame
Information
UI/Display
602
108

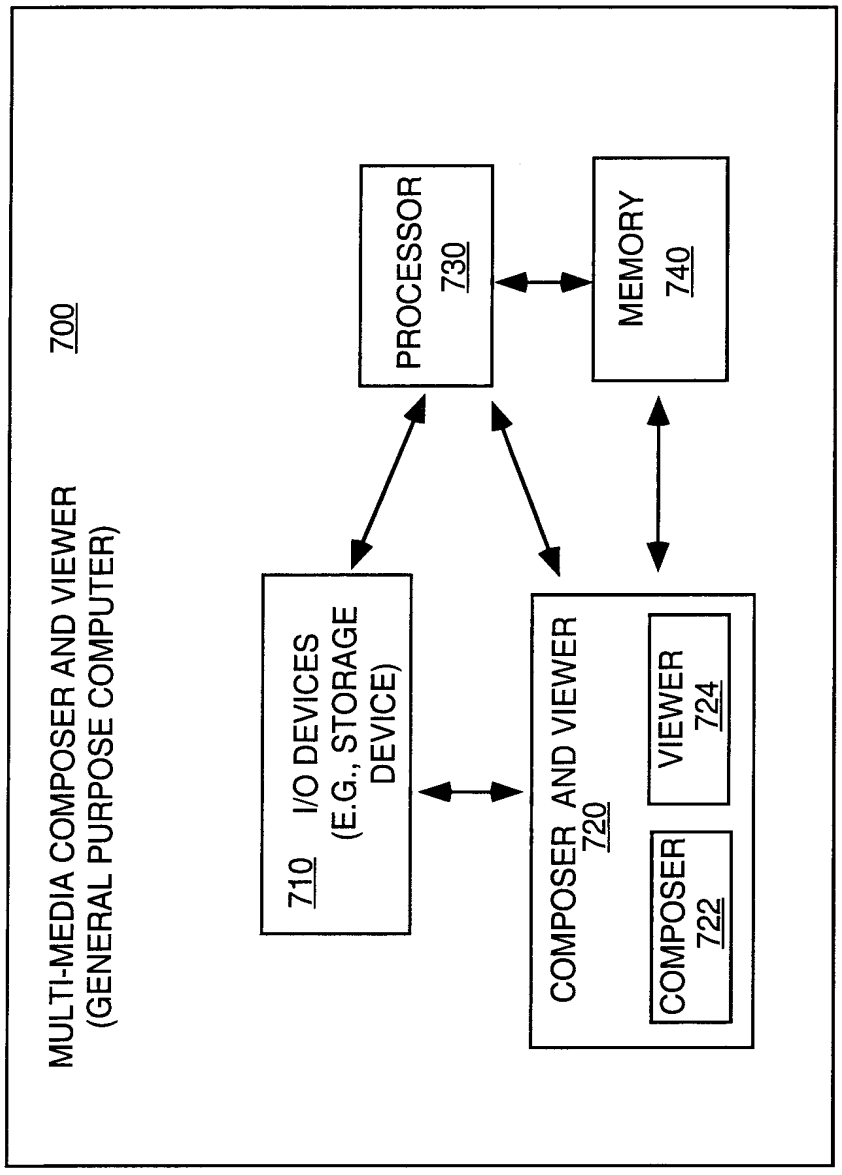

MULTI-MEDIA COMPOSER AND VIEWER
(GENERAL PURPOSE COMPUTER)
700
PROCESSOR
730
MEMORY
740
710 I/O DEVICES
(E.G., STORAGE
DEVICE)
COMPOSER AND VIEWER
720
VIEWER
724
COMPOSER
722

FIG. 7

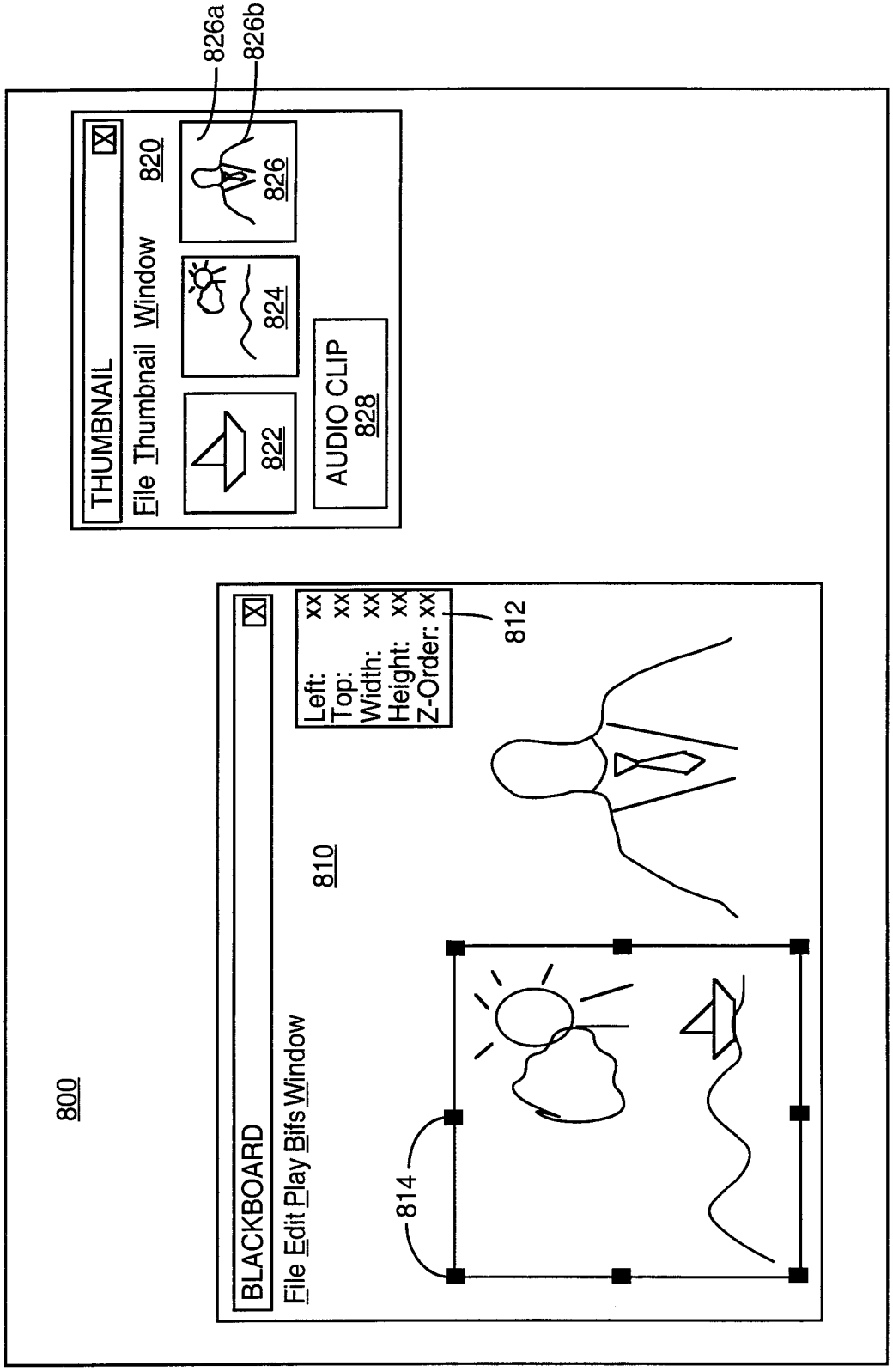

800
BLACKBOARD
File Edit Play Bifs Window
810
Left: xx
Top: xx
Width: xx
Height: xx
Z-Order: xx
812
814
THUMBNAIL
File Thumbnail Window
820
822
824
826
826a
826b
AUDIO CLIP
828

FIG. 8

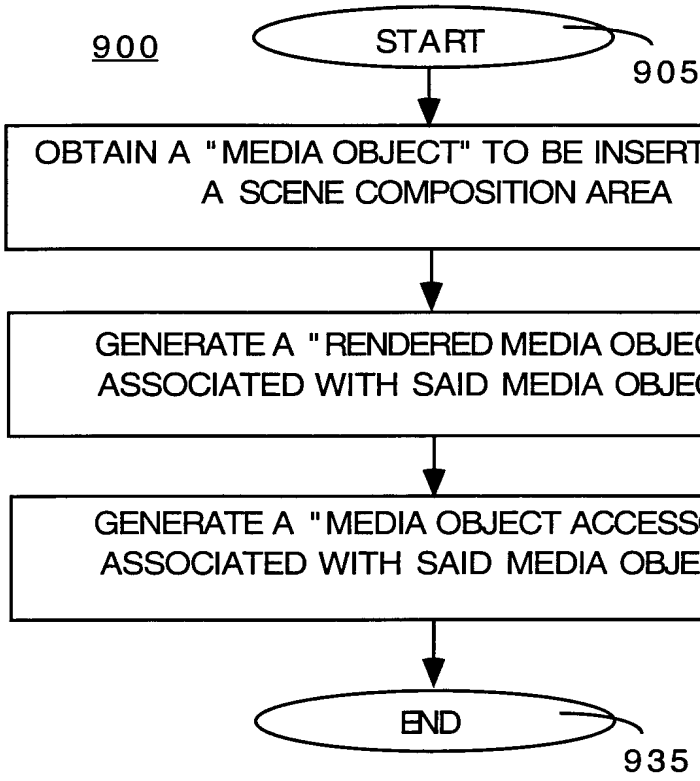

900
START
905
OBTAIN A "MEDIA OBJECT" TO BE INSERTED INTO A SCENE COMPOSITION AREA
910
GENERATE A "RENDERED MEDIA OBJECT" ASSOCIATED WITH SAID MEDIA OBJECT
920
GENERATE A "MEDIA OBJECT ACCESSOR" ASSOCIATED WITH SAID MEDIA OBJECT
930
END
935

FIG. 9

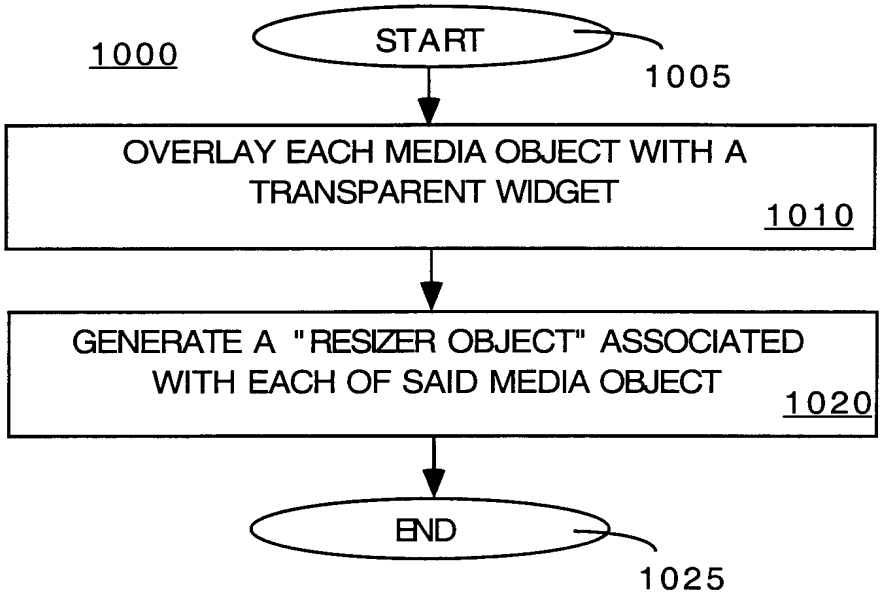

1000
START
1005
OVERLAY EACH MEDIA OBJECT WITH A TRANSPARENT WIDGET
1010
GENERATE A "RESIZER OBJECT" ASSOCIATED WITH EACH OF SAID MEDIA OBJECT
1020
END
1025

FIG. 10

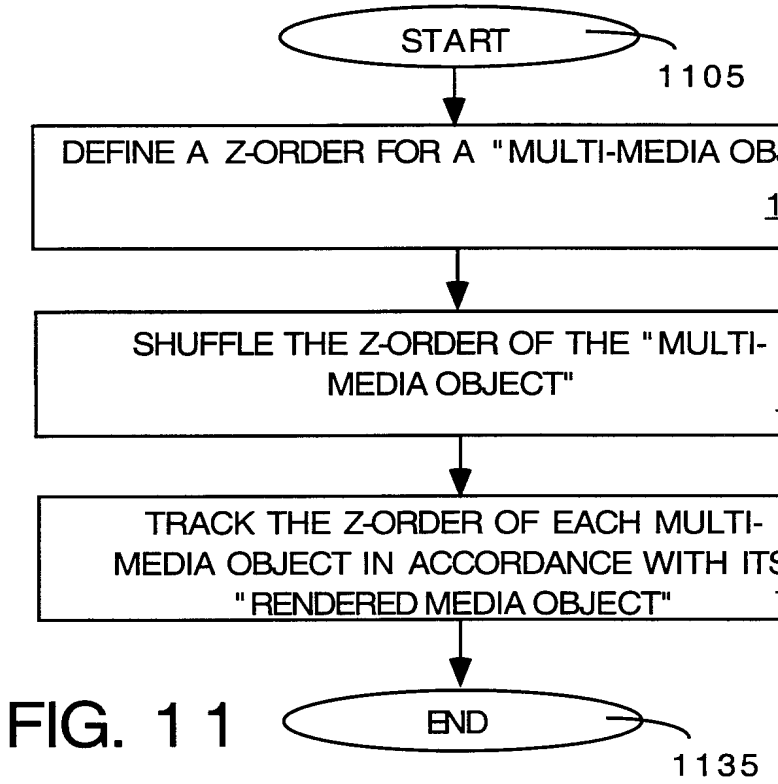

START
1105
DEFINE A Z-ORDER FOR A "MULTI-MEDIA OBJECT"
1110
SHUFFLE THE Z-ORDER OF THE "MULTI-MEDIA OBJECT"
1120
TRACK THE Z-ORDER OF EACH MULTI-MEDIA OBJECT IN ACCORDANCE WITH ITS "RENDERED MEDIA OBJECT"
1130
END
1135

FIG. 11

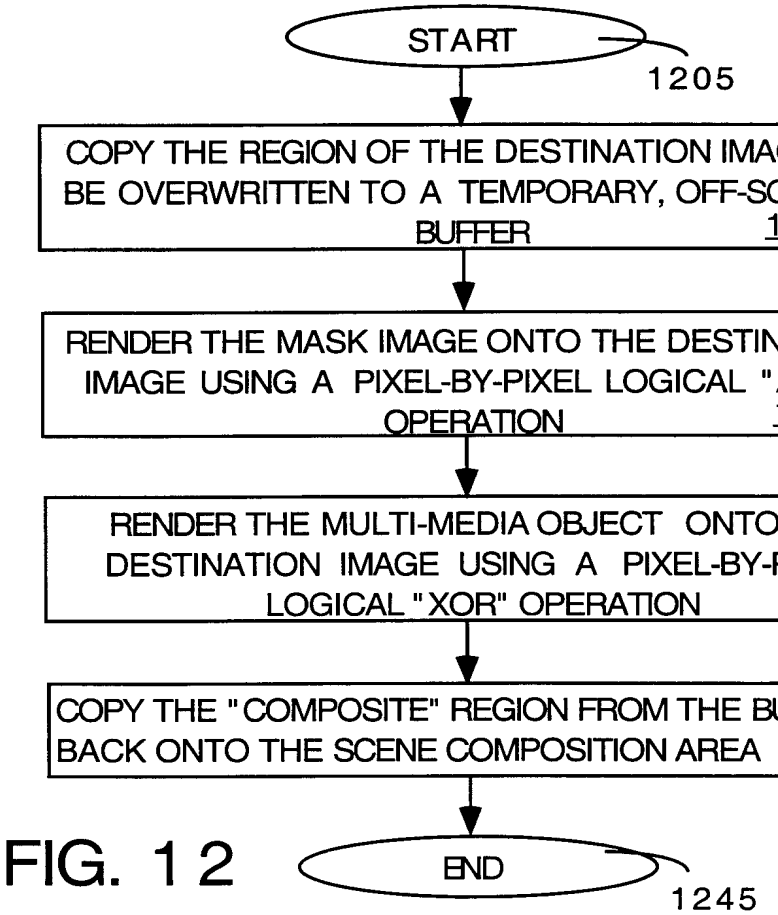

START
1205
COPY THE REGION OF THE DESTINATION IMAGE TO BE OVERWRITTEN TO A TEMPORARY, OFF-SCREEN, BUFFER
1210
RENDER THE MASK IMAGE ONTO THE DESTINATION IMAGE USING A PIXEL-BY-PIXEL LOGICAL "AND" OPERATION
1220
RENDER THE MULTI-MEDIA OBJECT ONTO THE DESTINATION IMAGE USING A PIXEL-BY-PIXEL LOGICAL "XOR" OPERATION
1230
COPY THE "COMPOSITE" REGION FROM THE BUFFER BACK ONTO THE SCENE COMPOSITION AREA
1240
END
1245

FIG. 12

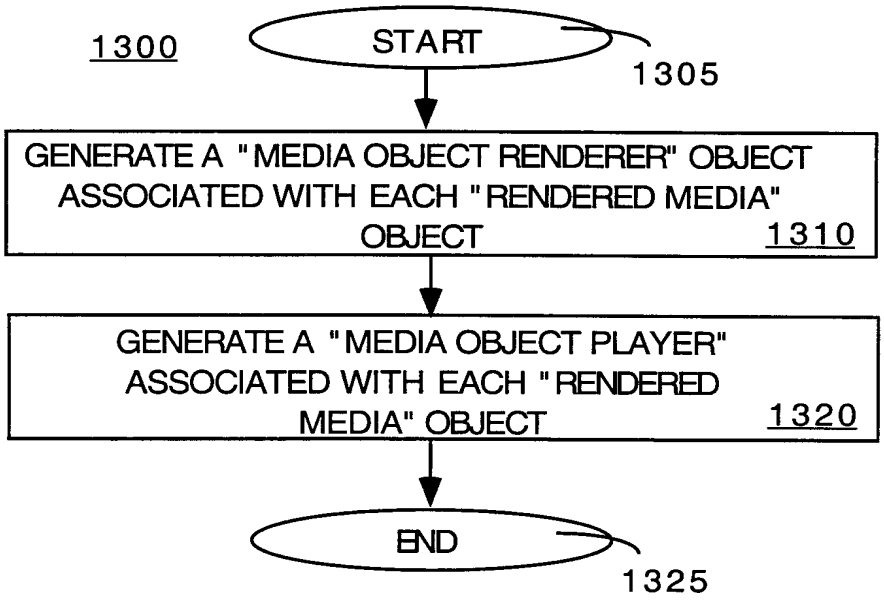

1300
START
1305
GENERATE A "MEDIA OBJECT RENDERER" OBJECT ASSOCIATED WITH EACH "RENDERED MEDIA" OBJECT
1310
GENERATE A "MEDIA OBJECT PLAYER" ASSOCIATED WITH EACH "RENDERED MEDIA" OBJECT
1320
END
1325

FIG. 13